Как ментально полюбить и начать писать тесты.

Рассматриваем на примере yii2, codeception

Докладчик: Волторнистый Артем

Опять?





Зачем я хотел писать тесты?

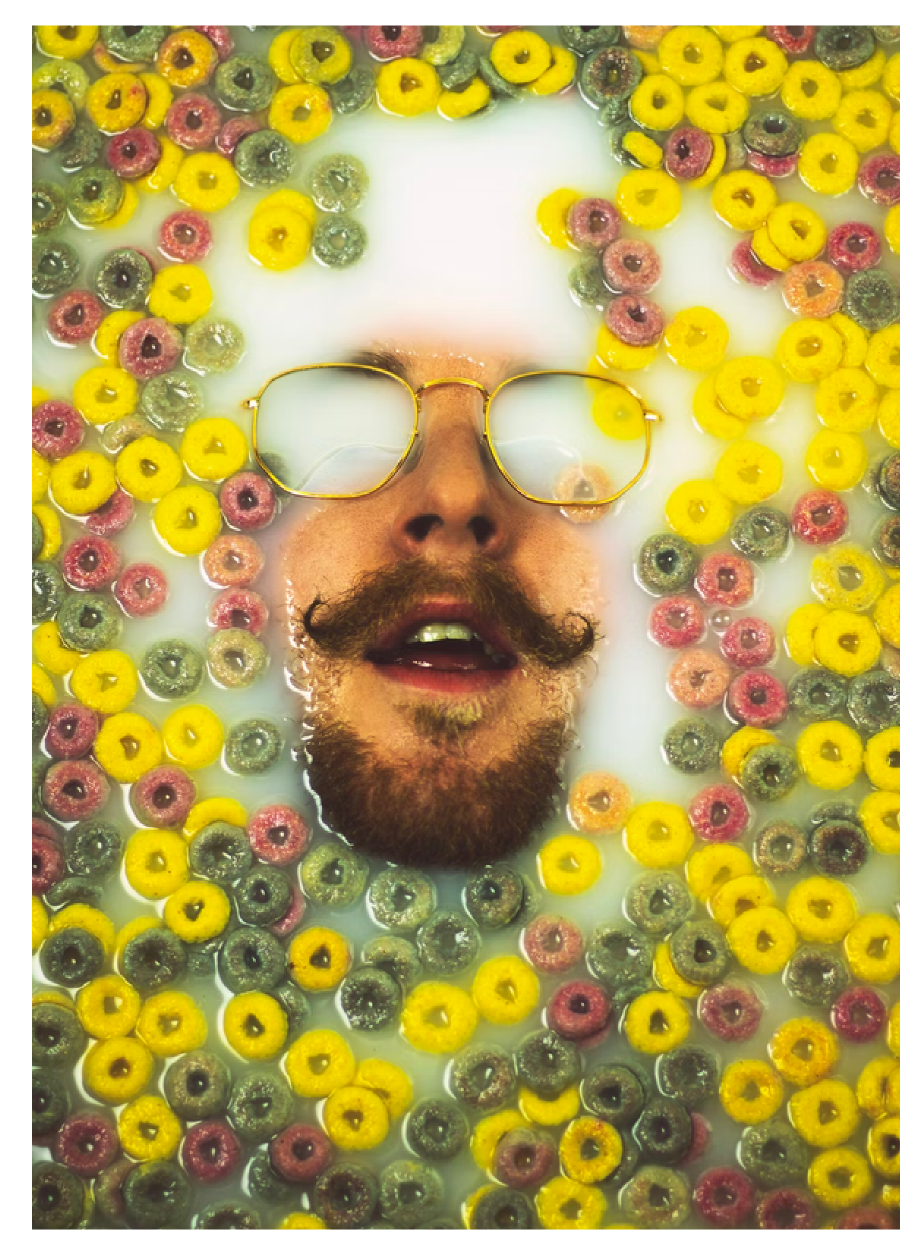


фото: @lukejonesdesign / unsplash.com

Код будет проверять

Наличие кода, который проверяет корректность моего кода после внесения изменений, позволит мне спать более спокойно.

Не надо все держать в голове

Я не могу помнить обо всех особенностях системы, я человек, а не робот, мозг периодически очищает память.



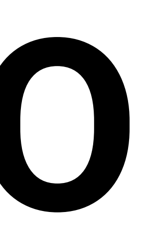




Б








билайн


php
PHP Russia
2022

Л





и

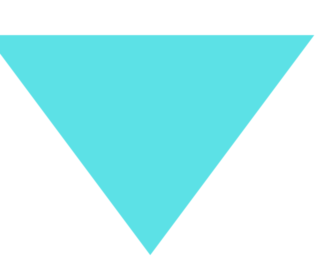




фото: @akshar_dave / unsplash.com







Я предположил, что если я протестировал класс для работы с базой, то я могу стабить его в других классах, которые его используют.

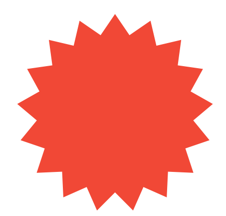


Стабы и моки

Внешнее взаимодействие принято стабить или мокать, то есть имитировать.







СТАБ — верни мне вот это в этом методе этого класса







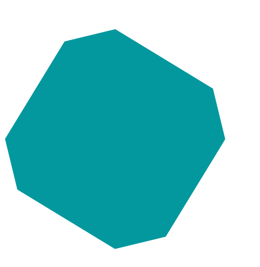
МОК — этот метод должен вернуть вот это и только один раз, не больше

Пример: хеш пароля





Не зависим от внешнего стенда, его режима работы, исключаем наше влияние на него, исключаем влияние сетевых событий: доступность, задержки.



Рассмотрим класс для тестирования запросов

```
// в методе класса unit-теста
$this->assertNull(
    (new UsersStore())->findById(1)
}
```





Теперь при тестировании другого класса — стабил базу

```
// содержимое метода testAuth в классе теста

$stub = Stub::make(UsersStore::class);
$stub->method('findById')->willReturn(null);

$service = new UserService($stub);

$I->expectThrowable(new DomainException("Пользователь не найден"), function () {

    $service->auth(new AuthUserForm(['login' => 'pushcat']));

});
```





Результат начинал меняться от параметров.

```
// содержимое метода testAuth в классе теста
$stub = Stub::make(UsersStore::class);
$stub->method('findById')->willReturn(null);
$stub->method('getById')->willReturn(null);
$stub->method('findByIdWithName')->willReturn(function ($id, $name) {
  if ($id == 2) {
     return ....
  }
  return null;
});
// ловим ошибку через expectThrowable
```





Какие типы ошибок я не мог поймать в стабах?

 Ошибки с первичными ключами

 Синтаксические ошибки

 Ошибки, связанные с функциями

....

БД у нас изначально пустая

Ее нужно как-то наполнять для тестов

Первоначальное состояние

После каждого теста система должна возвращаться в первоначальное состояние, чтобы исключить влияние других тестов друг на друга.

Добавим фикстуру

```
class UsersStoreTest extends Unit
{
    public function _fixtures() {
        return [
            'users' => [
                'class' => UsersFixture::class
            ]
        ]
    }

    public function testFindById() {
        $this->assertNull(
            (new UsersStore())->findById(1)
        )
    }
}
```





Фикстура

Заполняет нужные начальные данные для теста.

```
class UsersFixture {
    public $model = Users::class;
    public $file = '/path/to/file/';
}

// users.php

return [
    [
        'id' => 1,
        'name' => 'devnull',
        'status' => 1
    ]
];
```





Переносим данные из стабов в фикстуры

Да, я перешел на базу, в целом стало терпимо управлять наборами данных.

```
// класс Теста
    public function _fixtures() {
        return [
            'users' => [
                'class' => UsersFixture::class,
                'file' => 'default'
            ],
            'managers' => [...],
            'admin' => [...],
        ]
    }

    public function testAuth() {
        $service = new UserService(
            new UsersStore(
                Yii::$app->db
            )
        );
// expectThrowable
```





Когда подключалось не так много фикстур, все было хорошо.

Со временем кол-во начинало расти, и я не мог разобраться, что используется для теста:

 с каким статусом используются юзеры для теста?

 в файле фикстуры вижу поле type = 1 — что это? Шёл в модель...




Я хотел относительно быстро понять, что происходит в тесте, если давно их не открывал.

Например, это актуально было бы также и для новых разработчиков в команде.

Генерация данных

```
// метод testAuth класса теста

$user = Users::create()->assertSave();
$user->assignWithManager(
  $manager = Managers::create()->active()
);
$admin = Admin::create()->assertSave();
$admin->withPermission(Permission::profileRoute());

$holding = Holding::create("hold #1")->assertSave();
$holding->addManager($manager);

// вызов UserService->auth()
```





Генерация данных

Выглядеть стало более понятно, мне стало проще понимать, что происходит в тесте.

В классах генераторов можно внутри использовать Faker для рандомных данных.

Кстати, в Laravel для этого используют фабрики данных.

Но все равно было много тестов, частых правок....

Около года я писал тесты, а сложность увеличивалась...

Я понял, что что-то идет не так,
и удалил папку тестов.....
И тестировал ручками в браузере где-то полгода

Набирался сил, смотрел ютубчик, изучал материалы...

Для меня важными в итоге стали две концепции:

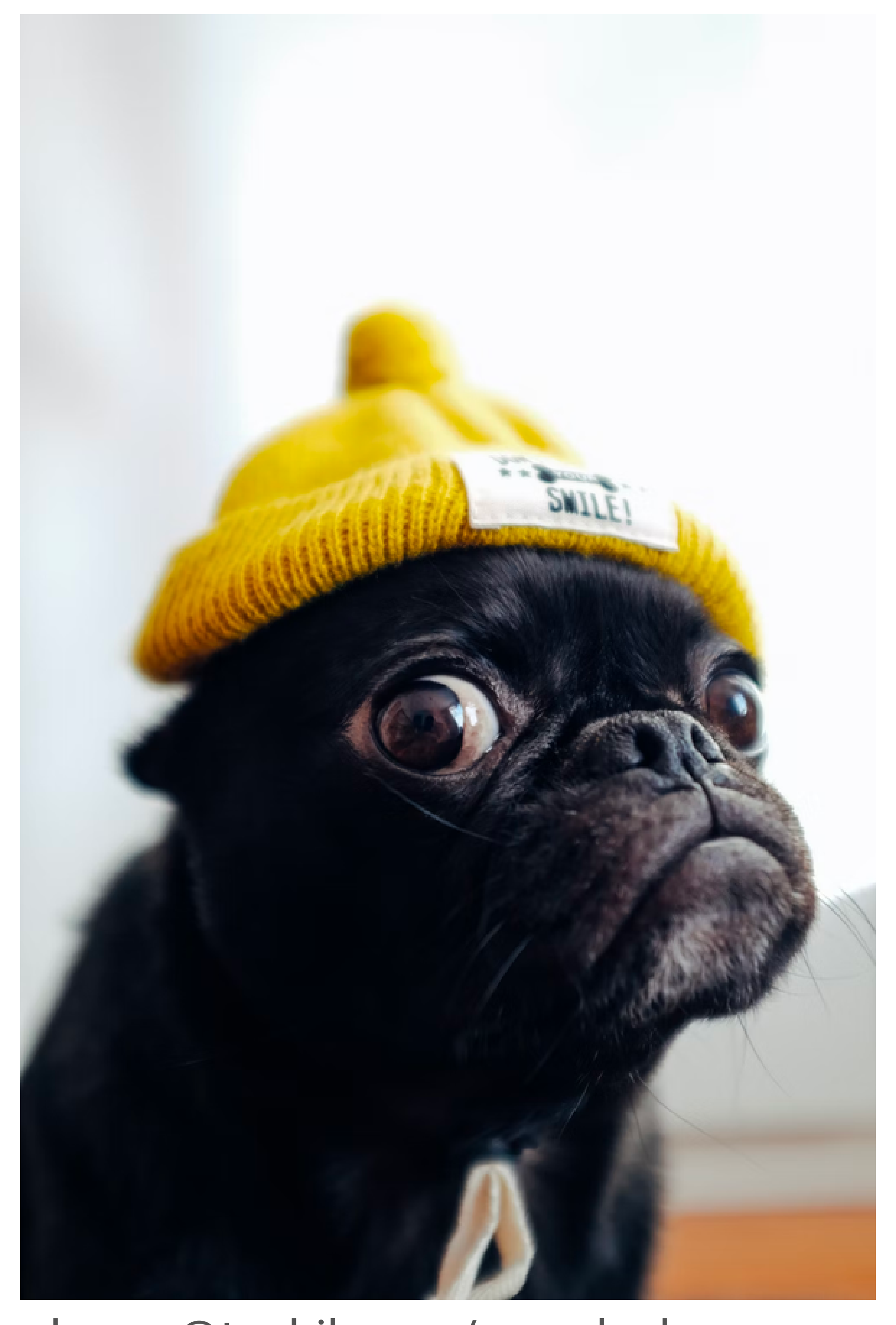


фото: @toshilepug / unsplash.com

Метод белого ящика
(white box)

Метод черного ящика
(black box)





Белый ящик

Проверяем форматы дат, наличие записей в бд, наличие джобов в очередях, отправку почты, ключи в кеше и т.д.

Что у нас все реализовывается корректно.

фото: @szamanm / unsplash.com





Черный ящик

Проверяем внешнее поведение:

- Поля формы можно заполнить, они подсвечиваются при ошибках, и кнопка оформления заказа активна при корректных полях и неактивна при некорректных.
- Например, после регистрации мы находимся на странице профиля и видим текст приветствия.



фото: @donovan_valdivia / unsplash.com







фото: @l_oan / unsplash.com



Как это происходит, через внешнее взаимодействие с другим сервером, работу с mysql или postgres — это всё нас не интересует.





Что выбрать?

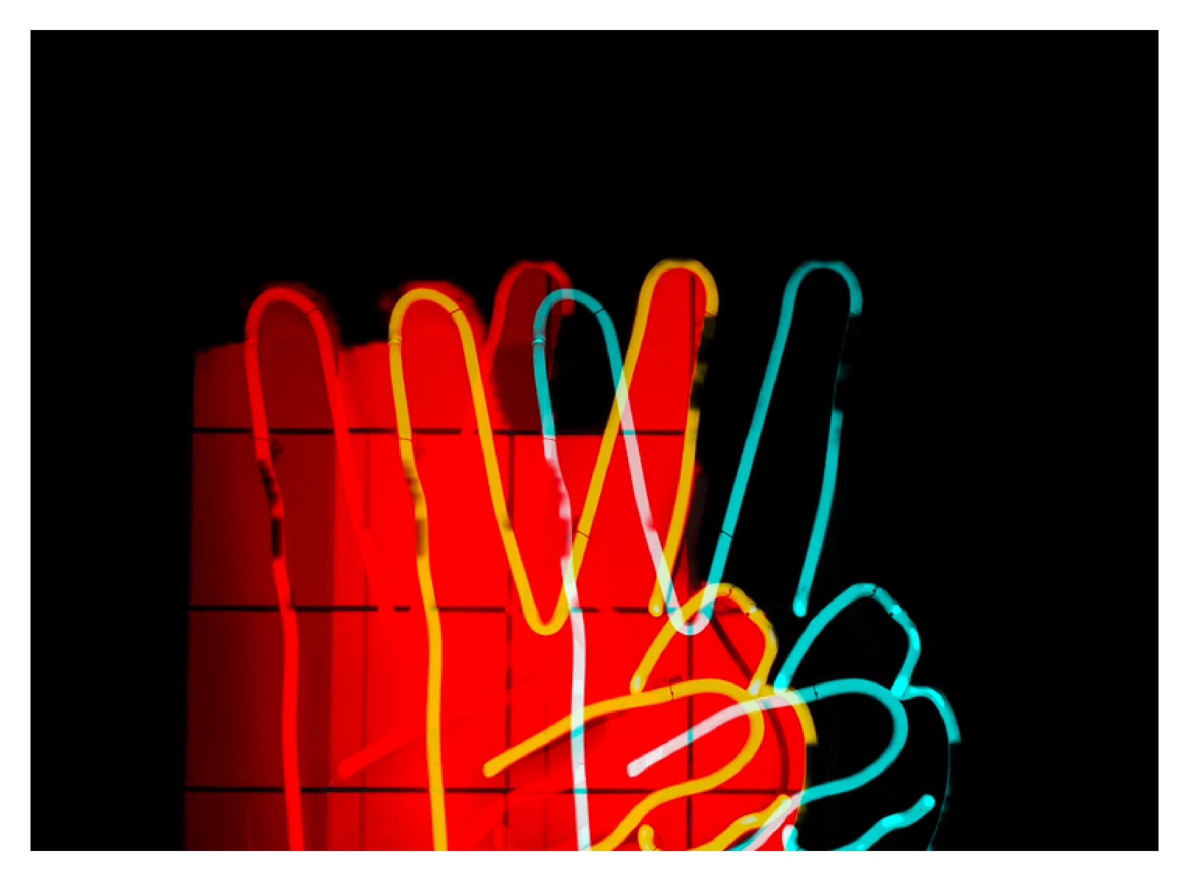


фото: @photoswithgabe / unsplash.com

После всей боли, что же я должен выбрать.....

Рассмотрим пример.

При тестировании методом черного ящика у меня были случаи, что, например, сам статус в бд оказался другим и у нас ошибка в if'е в контроллере, но все вроде работает.

Или, например, дата создания не установилась, а в гриде она выводится, потому что не было проверки на null.



фото: @jontyson / unsplash.com

"Здесь был php и оставил эту надпись, но он уже мёртв"

— golang





Давайте подойдем ближе к коду

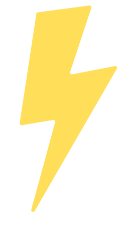
Как я мог это тестировать?



фото: @scentspiracy / unsplash.com

Первый способ

Писал тест на каждый класс, который используется в реализации наших бизнес-требований.

Какие это были классы

```php
class OrderController extends Controller
{
    public function actionCreate() {
        $form = new CreateOrderForm();
        if ($form->load($this->request->post()) && $form->validate()) {
            if ($form->save()) {
                return $this->redirect('success');
            }
        }
        return $this->render('index', compact('form'));
    }
}
```





Хотелось порой убить ноутбук



фото: @simplicity / unsplash.com





Второй способ

Post-запрос на урл создания заказа с последующей проверкой состояния:

- проверить наличие записи в бд
- проверить поля заказа
- проверить длину номера заказа
- проверить, что нас редиректнуло на некий урл заказа

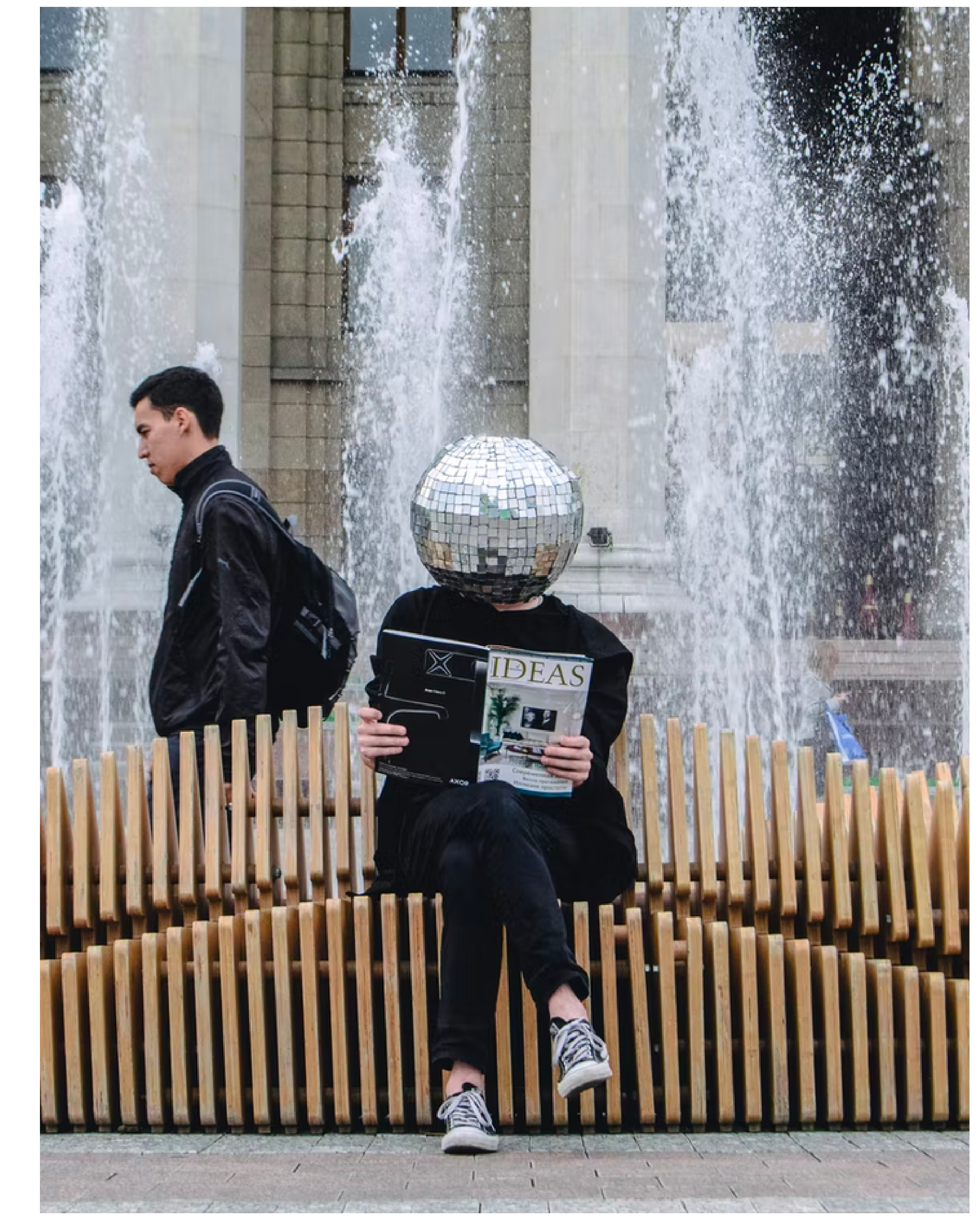


фото: @vale_zmeykov / unsplash.com


фото: @ameenfahmy / unsplash.com

Третий способ

Поднять браузер и покликать форму:

- форма заполняется
- поля подсвечиваются
- кнопка активна
- происходит редирект на нужный урл после успешного оформления

Приходит новое требование

Отправить уведомление в телегу менеджеру.

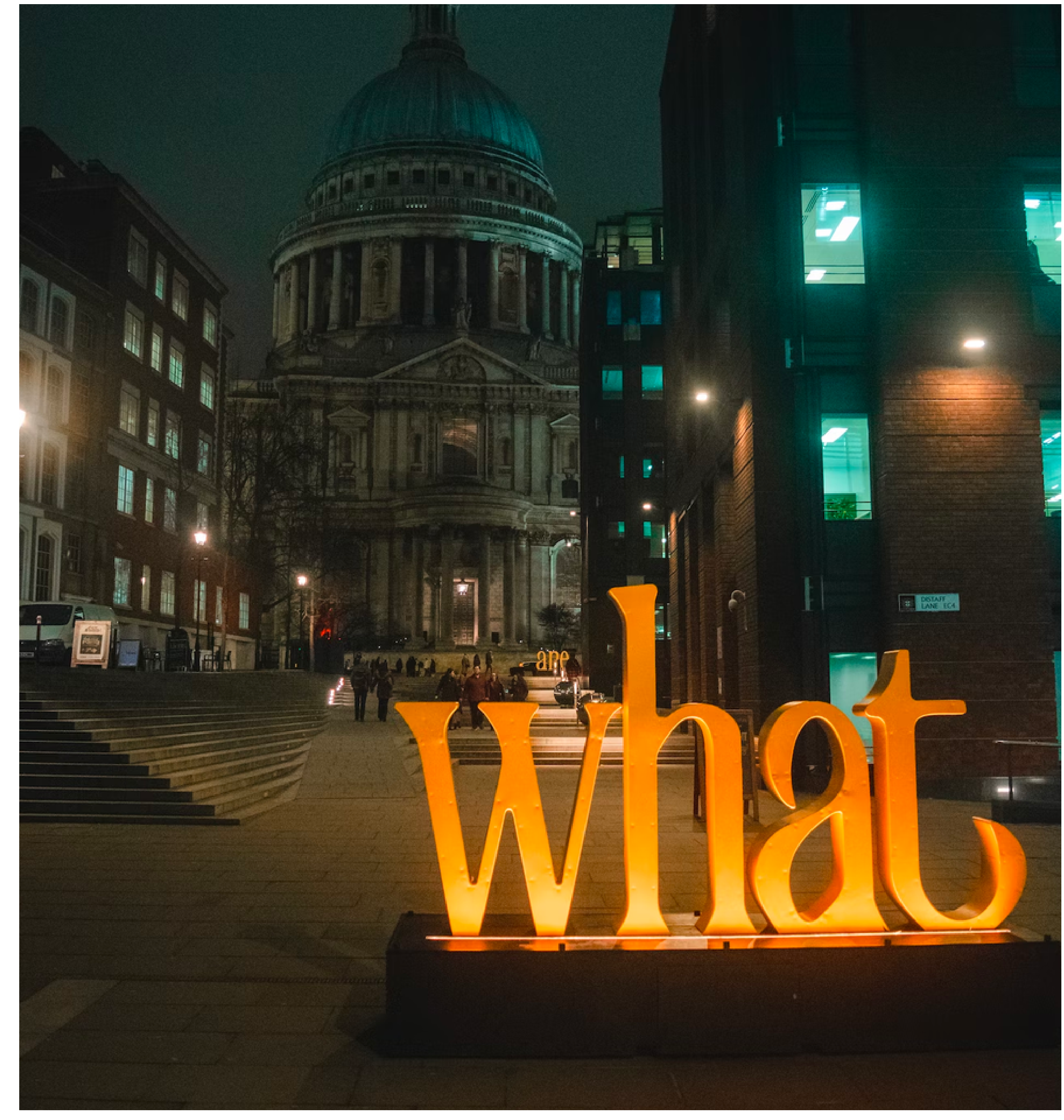


фото: @rhyskentish / unsplash.com

Решил сделать через очереди: кидаю джобу и при обработке кидаю уведомление менеджеру.

Что изменится?

В первом варианте дописываю тесты на новые классы

```
// OrderController
if ($form->save()) {

  Yii::$app->queue->push(
          new NewOrderJob(['id' => $form->getNewOrderId()])
  );
  Yii::$app->telega
          ->send(new Message($form->textMessage()));
  return $this->redirect('success');

}
// ...
```



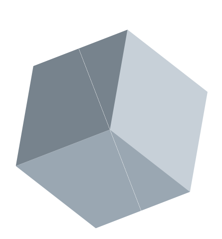


Во втором дописываю проверки


Создалась джоба


Итоговое сообщение телеграма имеет нужный текст в нужном формате





В третьем варианте, например, проверяю телеграм-менеджера или тестовый телеграм.



Рефакторинг





Есть код

```
class OrderController extends Controller
{
    public function actionCreate() {
        $form = new CreateOrderForm();
        if ($form->load($this->request->post()) && $form->validate()) {
            if ($form->save()) {
                $job = new NewOrderJob(['id' => $form->getNewOrderId()]);
                Yii::$app->rabbitQueue->push($job);
                return $this->redirect('success');
            }
        }
        return $this->render('index', compact('form'));
    }
}
```





Тимлид
сказал переписать наш контроллер по-новому
....

```php
class OrderController extends Controller
{
    public function actionCreate() {
        $form = new CreateOrderForm();
        if ($form->load($this->request->post()) && $form->validate()) {
            try {
                (new WithTransaction(
                    new OrderService(
                        new OrderStore(Yii::$app->db),
                        new Validator(),
                        new EventDispatcher(),
                        ......
                    )
                    new TransactionManager(Yii::$app->db)
                ))->create($form);
                $this->success("Заказ успешно оформлен")
                return $this->redirect('success');
            } catch (DomainException $e) {
                $this->error($e);
            }
        }
        return $this->render('index', compact('form'));
    }
}
```







Первый вариант

Я убивал почти все классы и все писал по-новой



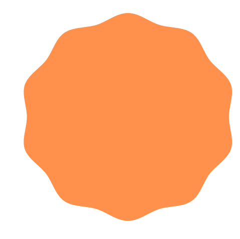
Второй и третий

Ничего не менял









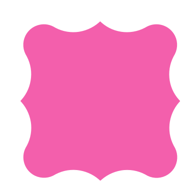
Я думал, зачем мне всё это, пишу тест на сервисный unit-класс и все тип-топ





Но тогда я не покрывал следующий случай:

WAF Handler	Trim Handler	/order/create /order/success	Error Handler



Используя второй и третий способы — покрываю.





Я сделал выводы:



- Речь про баланс, затраты, в том числе и моральные при первом написании тестов.
- Первым вариантом я повышаю стабильность работы, надежность.





Изучая более подробно, что я сделал, выяснил, что:



Первый

Unit-тесты



Второй

Интеграционное тестирование



Третий

Приемочное тестирование





Средним по затратам оказался второй вариант.
Минимум кода в тестах, удобство при рефакторинге.



Моя боль была в отсутствии баланса

С чем я столкнулся дальше...





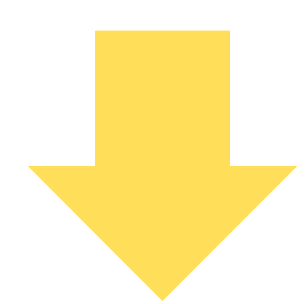
Я начал фантазировать дальше

Пользователь может создать заказ, менеджер его может подтвердить

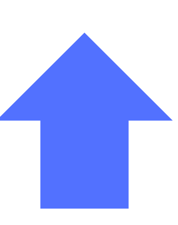
```
// метод create тесте OrderCest
$I->amOnPage(['/order/create']);
$I->seeResponseCode(200);
$I->submitForm('#create-order', [
    'CreateOrderForm' => [
        'name' => 'Pushkin',
        'phone' => '....'
        ....
  ]
]);
$I->cantSeeValidationErrors();
$I->seeInCurrentUrl(['/order/success']);
//....
```

Менеджер его может подтвердить

```
// метод confirm теста OrderCest
$I->amOnPage(['/order/create']);
$I->seeResponseCode(200);
$I->submitForm('#create-order', [
    'CreateOrderForm' => [
        'name' => 'Pushkin',
        'phone' => '....'
        ....
  ]
]);
$I->cantSeeValidationErrors();
$I->seeInCurrentUrl(['/order/success']);
// подтверждение
$I->amOnPage(['/admin/order/confirm']);
//....
```



По сути дальше я сталкиваюсь с проектированием кода, только уже для тестирования, чтобы было понятно, удобно.



Я ввел понятие DataSet и создал Page-классы (подсмотрел в сторону PageObject паттерна).





```php
class OrderPage {
    protected $tester;

    public function __construct(FunctionTester $tester) {
        $this->tester = $tester;
    }

    public function create(CreateOrderDataSet $data) {
        $this->tester->amOnPage(['/order/create']);
        $this->tester->seeResponseCode(200);
        $this->tester->submitForm('#create-order', $data->willSubmitData());
        $this->tester->cantSeeValidationErrors();
        $this->tester->seeInCurrentUrl(['/order/success']);
    }
}
```





```php
public function create(FunctionalTester $I, OrderPage $page) {
  //...
  $page->create(new CreateOrderDataSet(
    'name' => 'Pushkin',
    'phone' => '....'
     ....
  ));
  //....
}
```





```php
public function confirm(FunctionalTester $I, OrderPage $page) {
  $order = $page->create(new CreateOrderDataSet(
    'name' => 'Pushkin',
    'phone' => '....'
     ....
  ));

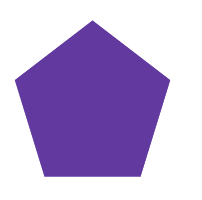
  // generate manager with permission

  $I->asUser($manager);

  $page->confirm(new ConfirmOrderDataSet(
    'id' => $order->id
  ));
  // asserts
  //.....
}
```







Снова фикстуры

Выглядит очень интересно, я могу просто дергать роуты и создавать нужные нам данные, но у меня была ситуация, когда это было слишком долго.

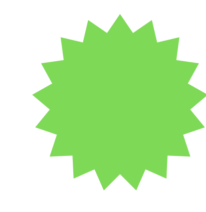
Проверить, отправился ли промокод после, например, оформления 1000-го заказа?







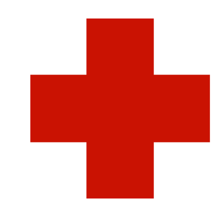
Я должен создать, потом подтвердить, потом смоделировать оплату — и так 1000 раз.











Будет очень долго, проще будет использовать фикстуру либо отдельный класс, в котором я буду 1000 раз создавать заказ, но потом создам дамп сгенерированных данных для будущего теста на скидку.

Давайте разберем тест с заказом более подробно





Кода пока еще нет

```
class OrderCest
{
    public function create(FunctionalTester $I) {

    }
}
```





```
class OrderCest
{
    public function create(FunctionalTester $I) {
        $I->amOnPage(['/order/create']);
    }
}
```

```php
class OrderCest
{
    public function create(FunctionalTester $I) {
        $I->amOnPage(['/order/create']);
        $I->seeResponseCode(200);
    }
}
```





Оформляем заказ

```php
//...
$I->amOnPage(['/order/create']);
$I->seeResponseCode(200);
$I->submitForm('#create-order', [
    'CreateOrderForm' => [
        'name' => 'Pushkin',
        'phone' => '....'
        ....
    ]
]);
$I->cantSeeValidationErrors();
$I->seeInCurrentUrl(['/order/success']);
```





```php
//...
$I->cantSeeValidationErrors();
$I->seeInCurrentUrl(['/order/success']);
$I->assertCount(1, $orders = Orders::find()->all());
$order = $orders[0];
$I->assertEquals('Pushkin', $order->name);
```

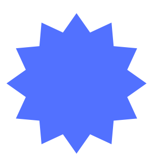
Я заметил, что тесты почти отображают бизнес-требования и я могу писать тест до кода. Записывать ТЗ под диктовку со слов менеджера?



фото: @jontyson / unsplash.com

Так это же вроде TDD?





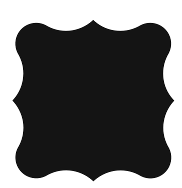
Я сперва описываю требования, потом под эти требования пишу код, чтобы он заработал.

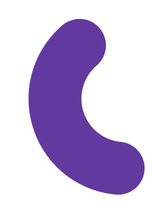



Когда приходит новое требование, я правлю тест, запускаю — он падает.

Я правлю код, тесты проходят.

Так же, когда я уточняю требования, так же добавляю заранее граничные случаи: заказов нет в БД, один заказ и множество заказов.





Если у меня случился баг, сперва пишу тест, который этот баг воспроизводит, и потом правлю код.





Когда приходит новое требование, я правлю тест, запускаю — он падает.

```
// OrderController
if ($form->save()) {

  Yii::$app->queue->push(
        new NewOrderJob(['id' => $form->getNewOrderId()])
  );
  Yii::$app->telega
        ->send(new Message($form->textMessage()));
  return $this->redirect('success');

}
// ...
```





- А как я мог проверить email?
- Как проверить, что ушло сообщение в очередь?
- Как проверить, что ушло сообщение в телегу?

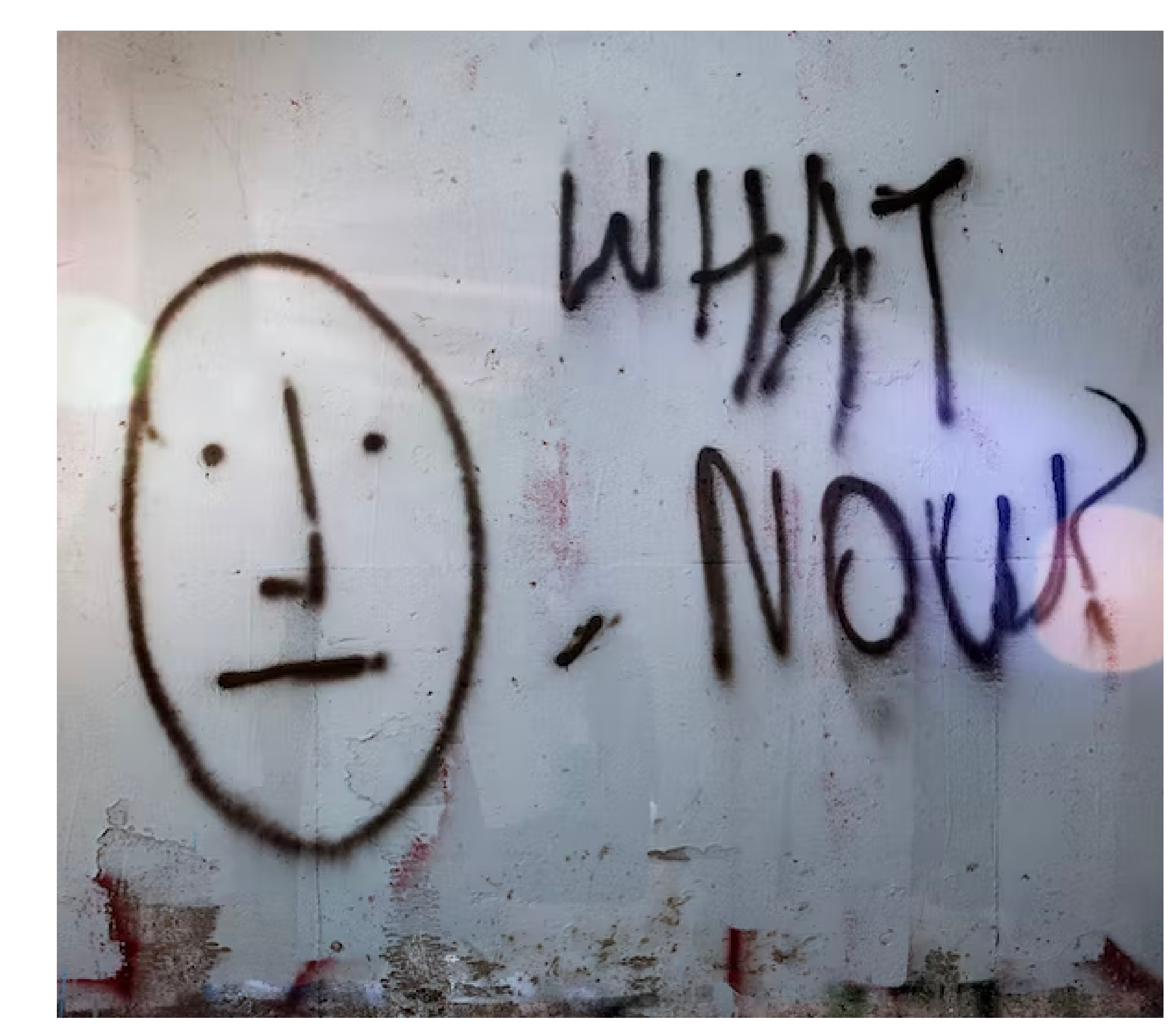


фото: @timmossholder / unsplash.com





Функции из коробки

```php
/**
 * @var MessageInterface $message
 */
$message = $I->grabLastSendEmail();
$I->assertArrayHasKey('sent@email.com', $message->getTo());
```





DI-контейнер

```php
Yii::$container->setSinglton(TelegramClient::class, function () {
    return new TelegramStubClient();
})

$I->amOnPage(['/order/create']);
$I->seeResponseCode(200);
//....
$I->assertCount(1, $messages = Yii::$container->get(TelegramClient::class)->messages()
$I->assertEquals('sender@hello.com', $messages[0]->getSender()->getEmail());
```





Для отлова ошибок:

```
Yii::$app->queue->on(Queue::EVENT_AFTER_ERROR, function(ExecEvent $event) {
    throw $event->error;
});
```

Сообщение ушло в очередь:

```
$queue->on(Queue::EVENT_AFTER_PUSH, function(PushEvent $event) use (&$count) {
        Debug::debug(["Job", VarDumper::dumpAsString($event->job, 3)]);
        if (null !== $count) {
            $count++;
        }
});
```





Пример проверки:

```php
/**
  * @var QueueFile $queue
  */
$queue = Yii::$app->queue_contracts;
$count = 0;
$I->attachEventQueueAfterPush($queue, $count);
//...
$I->assertEquals(1, $count);
$queue->run(0);
$I->assertEquals(2, $count);
```

Глобально:

```
namespace Helper\Module;

class Yii2Module extends Yii2 {
        public function _before(TestInterface $test):void {
                parent::_before($test);

                $queueNames = array_keys(require . 'config/queues.php');

                foreach ($queueNames as $queueName) {
                        $this->debugSection("Queues", "Clear $queueName");
                        $queue = Yii::$app->{$queueName};
                        $queue->clear();
                        $this->debugSection("Queues", "Set error handler $queueName");
                        $queue->on(Queue::EVENT_AFTER_ERROR, function(ExecEvent $event) {
                                throw $event->error;
                        });
                }

                $this->debugSection("Cache", "Clear cache");
                Yii::$app->cache->flush();
        }
}
```

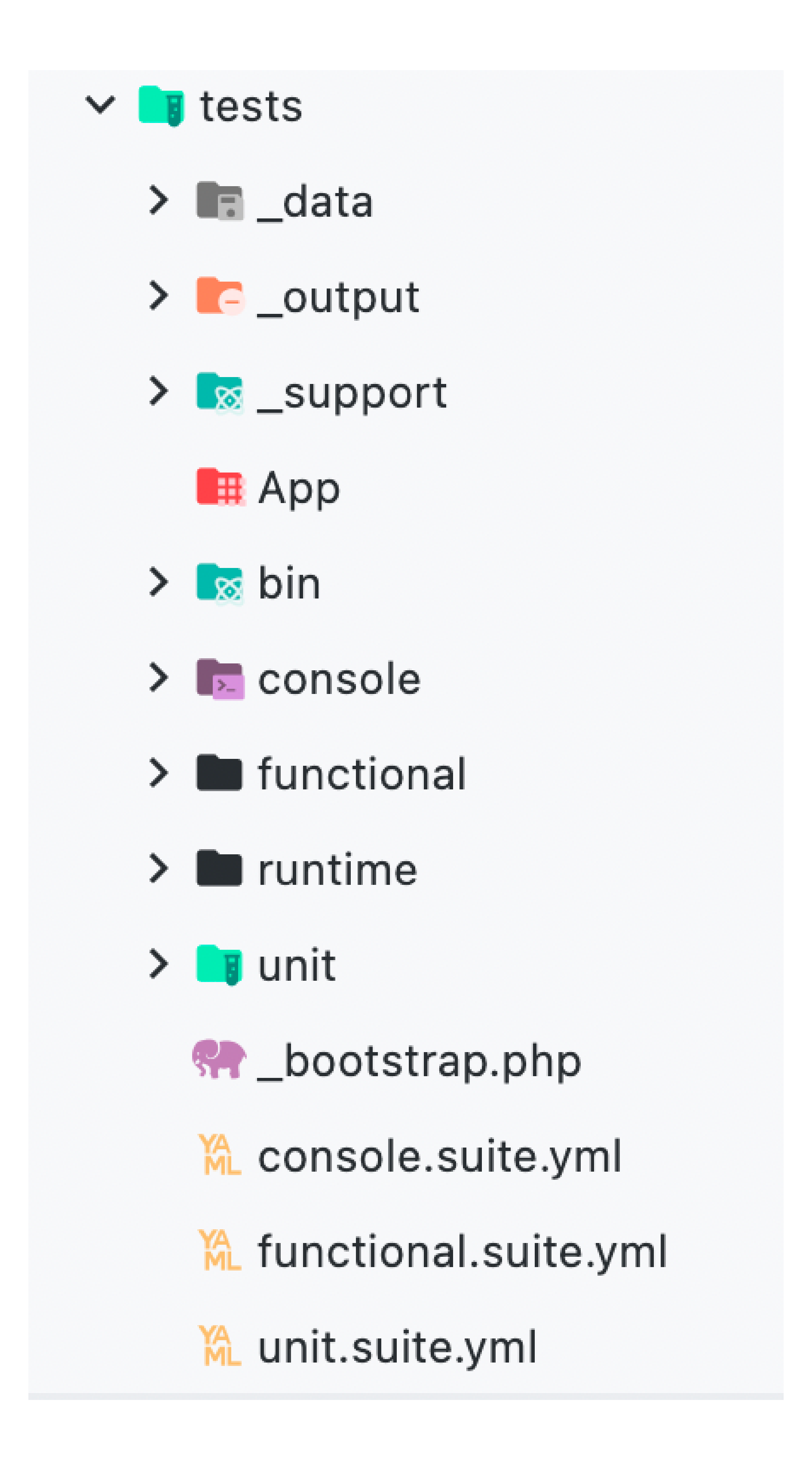




Codeception

Инструмент для тестирования

Так исторически сложилось, что я его стал использовать.







Пример конфига

Например, codeception.yml

```yaml
actor: Tester
bootstrap: _bootstrap.php
paths:
    tests: tests
    log: tests/_output
    data: tests/_data
    helpers: tests/_support
settings:
    memory_limit: 1024M
    colors: true
modules:
    config:
        Db:
            dsn: "mysql:host=db;dbname=test"
            user: "root"
            password: "password"
            populate: true
            cleanup: false
        Yii2:
            configFile: 'config/test.php'
```







фото: @jannerboy62 / unsplash.com

Когда я набил руку на тестах, я просто перестал о них говорить, а сразу на них закладывал время.

В среднем, они начали занимать 5-10% времени от самой задачи, но сколько плюсов.

При этом бизнесу могу объяснить важность, как и вам объяснил.

Когда мне надо было что-то сделать, я начинал думать, как мне это протестировать, иногда, когда это было сложно — я вспоминал, что могу протестировать руками : D

Дизайн кода

Когда я начал думать, как тестировать код, я понял, что можно изначально подумать, как будут называться наши функции и как их будут вызывать.

Пример с хешером, допустим, пусть он будет статическим.

```
class PasswordHasherTest extends \Codeception\Unit {

    public function testHash() {
        $hash = PasswordHasher::hash('password', 'sha251');
    }

}
```

Можно сразу описать граничные условия

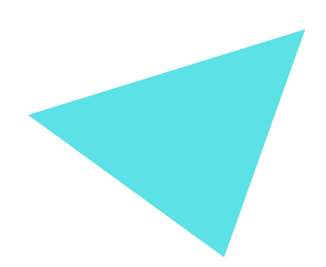
```
$hash = PasswordHasher::hash($pass = 'password', 'sha251');
$this->assertNotEmpty($hash);
$this->assertNotEquals($pass, $hash);
$this->assertLength(27, $hash);
```

А будет ли он удобен, если я захочу использовать нативную библиотеку?
Как это должно быть?

```php
PasswordHasher::hash('password', 'sha251', 'phpNativeLib');

PasswordHasher::hashByNativePhpLib('password', 'sha251');

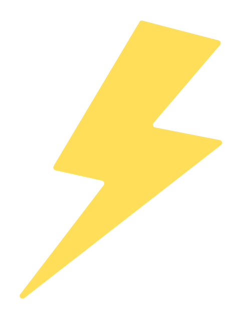
PasswordHasher::phpNative()->hash('password', 'sha251');
```



Другая команда использует DI-контейнер, будет ли наш хешер удобен?

Коллега решил замокать наш хешер, потому что он использует тяжелый алгоритм хеширования, которые выполняется около 20 сек — надо выставить более простой.

Дорогая операция.
Надо убедиться, что хешер в коде вызывается один раз.







Упс, а у нас статика.
Конечно, можно использовать спец. либы, но все же.
Думал я, статика ведь функция класса, а не объекта, и снова проектирование.

```
$hasher = Stub::make(PasswordHasher::class, [
    'hash' => Stub::once()
]);
$hash = (new User())->hash($hasher, 'password');
```





Это навело на мысль,
что разрабатывать "общий пакет" — не такая простая задача





На новом месте работы мои тесты были вне гита

Потом я их добавил в гит, потом в ci
в качестве ознакомительной
информации





Когда команда увидела падающие тесты при обновлении либы, — начали ощущать от них пользу.

Эта практика начала потихоньку распоространяться.



Также это было удобно на пет-проекте или закрытой бета-версии, когда я еще был заряжен и мне надо быстро накидать новый функционал, но при этом не ломать предыдущий.





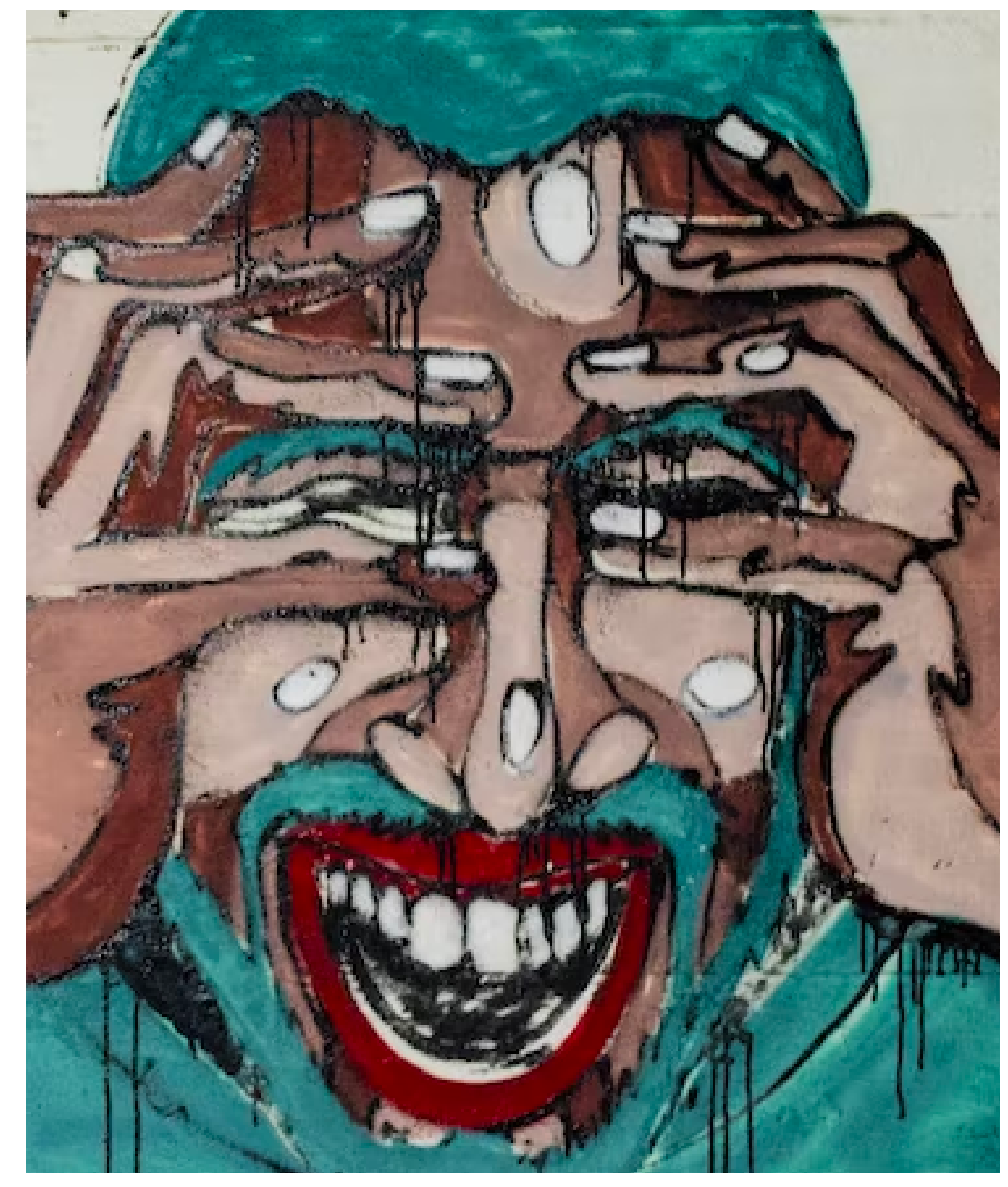


фото: @innernature / unsplash.com

А вдруг вас супруга/супруг попросят закодить, и у тебя баги или быстро надо будет накидывать функционал.

Будет неприятненько...



Рекомендации,
выводы







Удобство

Проще обновить ПО, фреймворк, язык
Собираем новую сборку в докере и запускаем тесты

В долгосрочной ускоряет разработку фич
Вы пишете новый функционал и при запуске тестов понимаете, что ничего не сломали. Меньше багов в бэклоге.

Тесты выступают почти как БТ
Проще понять, "как должно работать" приложение на нескольких тестах, чем составлять цепочку работы по всем файлам.


фото: @noahbuscher / unsplash.com



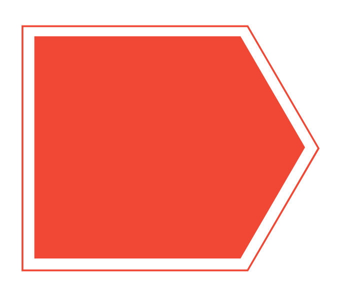
Писать тесты до кода можно
при определенной подготовке





Потребуется время, нужно попробовать





Это почти как привычка.
Если не писали тесты, надо
будет начать с малого,
например, с простых unit'ов.

Этот этап нужно пройти.

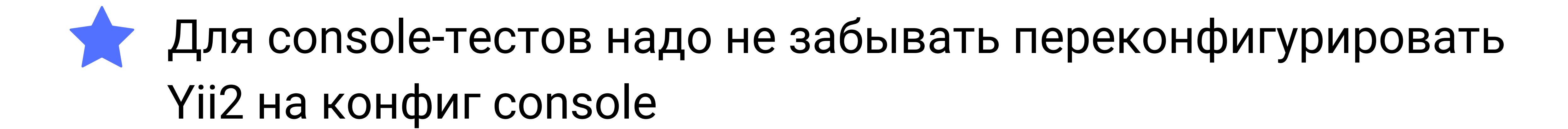


Начните с малого, установите codeception, настройте, создайте тест





Особенности codeception



- Для console-тестов надо не забывать переконфигурировать Yii2 на конфиг console

- При работе в функциональных тестах при переходе между контроллерами, bootstrap-файл лоадится только один раз

Особенности codeception

 При переключении с веба на консоль и обратно — возникают ошибки

 При тестировании апи через функциональные тесты, при появлении ошибки в апи в тестах будет отдаваться exception, а не конечный ответ апи. Мы не покрываем тест с error handler'ом. Лучше тестировать через приемочные, например, использовать PhpBrowser (curl-запросы).





Спасибо



Надеюсь, после моего доклада те, кто не хотел писать тесты, получат надежду и начнут потихоньку пробовать, а те, кто писал, найдут для себя что-то новое.



Оценить доклад

Волторнистый Артём

телеграм — @artemvolt

доклад — https://vk.cc/ciWXUU

